What can be decided locally without identifiers?

Pierre Fraigniaud[1]
CNRS and University Paris Diderot, France
pierre.fraigniaud@liafa.univ-paris-diderot.fr

Mika Göös
Department of Computer Science, University of Toronto, Canada
mika.goos@mail.utoronto.ca

Amos Korman[1]
CNRS and University Paris Diderot, France
amos.korman@liafa.univ-paris-diderot.fr

Jukka Suomela[2]
Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki, Finland
jukka.suomela@cs.helsinki.fi

Abstract. Do unique node identifiers help in deciding whether a network G has a prescribed property $\mathcal{P}$? We study this question in the context of *distributed local decision*, where the objective is to decide whether $G \in \mathcal{P}$ by having each node run a constant-time distributed decision algorithm. If $G \in \mathcal{P}$, all the nodes should output *yes*; if $G \notin \mathcal{P}$, at least one node should output *no*.

A recent work (Fraigniaud et al., OPODIS 2012) studied the role of identifiers in local decision and gave several conditions under which identifiers are not needed. In this article, we answer their original question. More than that, we do so under all combinations of the following two critical variations on the underlying model of distributed computing:

- (**B**): the size of the identifiers is *bounded* by a function of the size of the input network; as opposed to ($\neg$**B**): the identifiers are *unbounded*.
- (**C**): the nodes run a *computable* algorithm; as opposed to ($\neg$**C**): the nodes can compute any, possibly *uncomputable* function.

While it is easy to see that under ($\neg$**B**, $\neg$**C**) identifiers are not needed, we show that under all other combinations there are properties that can be decided locally if and only if identifiers are present. Our constructions use ideas from classical computability theory.

Keywords: Distributed complexity; local decision; identifiers; computability theory.

[1]Additional support from the ANR projects DISPLEXITY, and from the INRIA project GANG.
[2]This work was supported in part by the Academy of Finland, Grants 132380 and 252018, and by the Research Funds of the University of Helsinki.

1 Introduction

In this work we ask and answer a simple question: *Do we need unique node identifiers when locally deciding a graph property?* While this question is a natural one, our answers are somewhat artificial—but only necessarily so.

Local decision. A property of graphs $\mathcal{P}$ is *locally decidable* if there is a distributed algorithm A (in the usual $\mathcal{LOCAL}$ model; see Section 1.2) with a constant running time $t = O(1)$ that when run on a graph G can decide whether $G \in \mathcal{P}$ in the following sense:

- if $G \in \mathcal{P}$, then A outputs *yes* on every node of G, and
- if $G \notin \mathcal{P}$, then A outputs *no* on at least one node of G.

Here, the output of A on a node $v \in V(G)$ can only depend on the information that is available to within t steps of v in G. This includes not only the radius-t neighbourhood topology around v, but also—as is often assumed—numerical identifiers $\mathrm{Id}(u)$ for each node u in the neighbourhood. The assignment $\mathrm{Id}\colon V(G) \to \mathbb{N}$ is one-to-one.

Do we need identifiers? Recently, Fraigniaud et al. [5] asked whether it makes any difference in this context to have A's output depend on the identifiers $\mathrm{Id}(v)$. After all, whether G has the property $\mathcal{P}$ or not does not depend on how the nodes of G are labelled with identifiers, and moreover, the usual challenge of *local symmetry breaking* does not arise in the context of decision problems.

Indeed, they conjectured that for any local algorithm A that decides a property $\mathcal{P}$ there is an equivalent *Id-oblivious* local algorithm A^* that decides $\mathcal{P}$ and that does not use identifiers in the sense that the output of A^* on a node $v \in V(G)$ does not change if we reassign the identifiers, i.e., $A^*(G, \mathrm{Id}, v) = A^*(G, \mathrm{Id}', v)$ for any two assignments $\mathrm{Id}, \mathrm{Id}'\colon V(G) \to \mathbb{N}$.

In this work, we disprove the conjecture. We show that there are graph properties whose local decision requires the output of a constant-time algorithm to depend on the identifier assignment—if the details of the underlying model of distributed computation are set up in a particular way.

Assumptions. To understand what our question entails on a technical level, we need to make explicit two critical assumptions about the model of computing.

Size of identifiers. It is commonly assumed that the identifiers are given as $O(\log n)$-bit labels in a graph with n nodes. It is debatable whether it is natural to require bounded identifiers in our case of constant-time algorithms; in any case, we consider both alternatives:

- (**B**) The size of identifiers is *bounded* by a function of n.
- (¬**B**) The size of identifiers is *unbounded*.

Note that, since a local algorithm operates on a graph component-wise, there is no distinction between (**B**) and (¬**B**) if we allow all disconnected graphs as input: in either case there will be no bound on $\mathrm{Id}(v)$ as a function of the size of v's component. Thus, in what follows, we work under the promise that the input graph is connected. We will show that whether identifiers help in local decision depends on which of the assumptions (**B**) or (¬**B**) we adopt.

Computability. Second, should we restrict the power of local computations? We have two alternatives:

- (**C**) The nodes run a *computable* algorithm.
- (¬**C**) The nodes can compute any function, possibly *uncomputable*.

For many questions in distributed computing, the distinction between (**C**) and (¬**C**) is inconsequential and not interesting. However, we will show that whether identifiers help in local decision depends on which of the assumptions (**C**) or (¬**C**) we adopt.

Id-oblivious simulation. Our results are best motivated by the observation that identifiers are not needed under (¬**B**, ¬**C**). Indeed, if A is a t-time algorithm deciding a property $\mathcal{P}$, we can simulate A by an Id-oblivious t-time algorithm A^*.

> Id-oblivious simulation A^*: For each local neighbourhood (G', v), $G' \subseteq G$, algorithm A^* checks whether there is a local assignment $\mathrm{Id}' \colon V(G') \to \mathbb{N}$ that makes the output $A(G', \mathrm{Id}', v)$ be *no*. If such an assignment exists, we let A^* output *no* on v, too; otherwise, we let A^* output *yes* on v.

We first note that, even though A^* is well-defined, it is not obvious how to compute it, since finding out whether Id' exists might involve an exhaustive search over an infinite domain. For example, even if A was computable to start with, our A^* is now deciding, a priori, a *computably enumerable* predicate. However, under (¬**C**), this is not a problem.

To see that A^* correctly decides $\mathcal{P}$, we note that A^* outputs *no* on some node in G, if and only if there is some global assignment $\mathrm{Id} \colon V(G) \to \mathbb{N}$ (i.e., extension of Id') that makes A output *no* on some node. The identifiers in the assignment Id may be very large, but under (¬**B**) this is not a problem. Thus, (G, Id) is a valid input for A, and the correctness of A^* now follows from that of A.

Our main result in this work is showing that there is no general Id-oblivious simulation in case one of the assumptions (**B**) or (**C**) is imposed.

1.1 Our results

We show that identifiers are necessary in local decision under (**B**), and under (**C**).

Theorem 1. *Assume* (**B**) *or* (**C**). *There is a locally decidable property $\mathcal{P}$ that cannot be decided with an Id-oblivious local algorithm.*

In particular, this separates the classes LD and LD^* that were previously conjectured to be equal under (¬**B**, **C**) by Fraigniaud et al. [5]. Here, LD is the class of locally decidable properties, and $\mathsf{LD}^* \subseteq \mathsf{LD}$ is the class of properties decidable with an Id-oblivious local algorithm.

We prove the separation $\mathsf{LD}^* \neq \mathsf{LD}$ assuming (**B**, ¬**C**) in Section 2, and again assuming (**C**) in Section 3. For the latter, more involved separation, we end up using ideas from classical (sequential) computability theory. The use of these techniques should not come as a surprise given that $\mathsf{LD}^* = \mathsf{LD}$ under (¬**B**, ¬**C**) as discussed above. We collect the relationships between LD^* and LD in the following table:

	(**C**)	(¬**C**)	
(**B**)	$\neq$	$\neq$	→ Section 2
(¬**B**)	$\neq$	$=$	

↳ Section 3

Finally, we note that the property $\mathcal{P}$ that witnesses $\mathsf{LD} \neq \mathsf{LD}^*$ under (**C**) becomes decidable with an Id-oblivious algorithm if we allow *randomness*.

Corollary 1. *Property $\mathcal{P}$ can be decided (w.h.p.) with an Id-oblivious randomised local algorithm.*

Randomised local decision was previously studied by Fraigniaud et al. [3, 6]. The corollary above indicates, in particular, that in the Id-oblivious model, the threshold result [3, Theorem 3.3] that pertains to hereditary languages does not hold if we consider all languages.

1.2 Local decision in the $\mathcal{LOCAL}$ model

A *labelled graph* is a pair $(G, \boldsymbol{x})$, where $G = (V(G), E(G))$ is a simple undirected graph and function $\boldsymbol{x}$ associates a *label* or a *local input*, denoted $\boldsymbol{x}(v)$, with each node $v \in V(G)$.

A *labelled graph property* is a collection $\mathcal{P}$ of labelled graphs that is invariant under graph isomorphism. That is, if $(G, \boldsymbol{x}) \in \mathcal{P}$, and $(G', \boldsymbol{x}')$ is isomorphic to $(G, \boldsymbol{x})$, then $(G', \boldsymbol{x}') \in \mathcal{P}$. Examples of labelled graph properties include the following:

- "proper 3-colouring": $(G, \boldsymbol{x}) \in \mathcal{P}$ if $\boldsymbol{x}$ is a proper 3-colouring of G,
- "maximal independent set": $(G, \boldsymbol{x}) \in \mathcal{P}$ if the nodes with $\boldsymbol{x}(v) = 1$ form a maximal independent set in G,
- "planar graphs": $(G, \boldsymbol{x}) \in \mathcal{P}$ if G is a planar graph (and $\boldsymbol{x}$ is arbitrary).

In particular, all graph properties can be interpreted as labelled graph properties. If $\mathcal{P}$ is a property, we say that any pair $(G, \boldsymbol{x}) \in \mathcal{P}$ is a *yes-instance* and any pair $(G, \boldsymbol{x}) \notin \mathcal{P}$ is a *no-instance*.

An *input* is a triple $(G, \boldsymbol{x}, \mathrm{Id})$, where $(G, \boldsymbol{x})$ is a labelled graph and $\mathrm{Id}\colon V(G) \to \mathbb{N}$ is a one-to-one function. We say that $\mathrm{Id}(v)$ is the *unique identifier* of node $v \in V(G)$.

Local algorithms. Let $B(v, t) \subseteq V(G)$ consist of the nodes that are within distance t from v in graph G. We write $(G, \boldsymbol{x}, \mathrm{Id}) \restriction B(v, t)$ for the restriction of the structure $(G, \boldsymbol{x}, \mathrm{Id})$ to $B(v, t)$.

Let A be a function that associates a *local output* $A(G, \boldsymbol{x}, \mathrm{Id}, v) \in \{\mathit{yes}, \mathit{no}\}$ with each node $v \in V$ for any input $(G, \boldsymbol{x}, \mathrm{Id})$. We say that A is a *local algorithm* with local horizon t if $A(G, \boldsymbol{x}, \mathrm{Id}, v) = A(G', \boldsymbol{x}', \mathrm{Id}', v)$ whenever $(G, \boldsymbol{x}, \mathrm{Id}) \restriction B(v, t) = (G', \boldsymbol{x}', \mathrm{Id}') \restriction B(v, t)$. That is, in a local algorithm the local output of node v depends only on the information that is available in the radius-t neighbourhood of node v.

We say that local algorithm A is *Id-oblivious* if $A(G, \boldsymbol{x}, \mathrm{Id}, v) = A(G, \boldsymbol{x}, \mathrm{Id}', v)$ for any two assignments $\mathrm{Id}, \mathrm{Id}'\colon V(G) \to \mathbb{N}$. That is, renumbering the identifiers does not change the output of an Id-oblivious algorithm. Indeed, we may write the output simply as $A(G, \boldsymbol{x}, v)$.

While in the above description we have specified a local algorithm as a function that maps local neighbourhoods to local outputs, we could equally well specify a local algorithm from the perspective of networked state machines that exchange messages with each other: graph G is the structure of the network, each node is a computer, each edge is a communication link, all nodes run the same algorithm, and a node $v \in V(G)$ initially knows only $\boldsymbol{x}(v)$ and $\mathrm{Id}(v)$. In essence, a local algorithm with local horizon t is equivalent to a distributed algorithm that runs in $t \pm 1$ synchronous communication rounds in the $\mathcal{LOCAL}$ model [16, 20].

Assumptions. Under assumption (**B**), we assume that there is a function f such that $\mathrm{Id}(v) < f(|V(G)|)$ for any input $(G, \boldsymbol{x}, \mathrm{Id})$.

Under assumption (**C**), we require that local algorithm A is a computable function of the local neighbourhood. Put otherwise, we require that there is a Turing machine M_A such that for any input $(G, \boldsymbol{x}, \mathrm{Id})$ and any node $v \in G$, given a string that encodes node v and the local neighbourhood $(G, \boldsymbol{x}, \mathrm{Id}) \restriction B(v, t)$, machine M_A halts and outputs $A(G, \boldsymbol{x}, \mathrm{Id}, v)$.

Local decision. Local algorithm A *decides* a property $\mathcal{P}$ if the following holds for any input $(G, \boldsymbol{x}, \mathrm{Id})$:

- if $(G, \boldsymbol{x}) \in \mathcal{P}$, then $A(G, \boldsymbol{x}, \mathrm{Id}, v) = \mathit{yes}$ for all $v \in V(G)$,
- if $(G, \boldsymbol{x}) \notin \mathcal{P}$, then $A(G, \boldsymbol{x}, \mathrm{Id}, v) = \mathit{no}$ for at least one $v \in V(G)$.

If there is a local algorithm that decides $\mathcal{P}$, we say that $\mathcal{P}$ is in class LD. If there is an Id-oblivious local algorithm that decides $\mathcal{P}$, we say that $\mathcal{P}$ is in class LD^*.

Promise problems. While our constructions do not make use of promise problems, we will refer to them in some introductory examples. If we say that we have *promise* $\mathcal{P}'$, then we are only interested in inputs $(G, \boldsymbol{x}, \mathrm{Id})$ with $(G, \boldsymbol{x}) \in \mathcal{P}'$.

In particular, if $(G, \boldsymbol{x}, \mathrm{Id})$ is an input that violates the promise, we do not put any requirements on $A(G, \boldsymbol{x}, \mathrm{Id}, v)$. Even if we work under assumption (**C**), we do not require that machine M_A halts for inputs that violate the promise. Put otherwise, A can be a partial function, undefined for inputs that violate the promise.

1.3 Related work

The question of how to locally decide (or verify) languages has been gaining attention in recent years [1, 3, 5, 8, 11–14]. Inspired by traditional computational complexity theory, Fraigniaud et al. [3] suggested that the study of decision problems may provide new structural insights also in the distributed computing setting. While the original focus was on the $\mathcal{LOCAL}$ model, recent work has taken the first steps towards a computational complexity theory in various other contexts of distributed computing [2, 4, 7].

Local decision. The classes LD, NLD and BPLD defined by Fraigniaud et al. [3] are the distributed analogues of the classes P, NP and BPP, respectively. The paper [3] provides structural results, develops a notion of local reduction, and establishes completeness results. One of the main results is that, for a large class of languages, called *hereditary languages*, there exists a sharp threshold for randomisation, above which randomisation does not help.

Identifiers and local decision. More recently, Fraigniaud et al. [5] defined the *Id-oblivious* model, and the corresponding class of languages LD^*, aiming to understand better the role of identities in local decision. They also conjectured that $\mathsf{LD}^* = \mathsf{LD}$. Informally, the conjecture states that for constant time computations, identities do not play any role except for allowing nodes to identify their local neighbourhoods.

Several positive evidences where given supporting this conjecture [5]. Specifically, it is shown that $\mathsf{LD}^* = \mathsf{LD}$ holds for hereditary languages and languages defined on paths, with a finite set of input values. Moreover, it was shown that equality holds in the non-deterministic setting, i.e., $\mathsf{NLD}^* = \mathsf{NLD}$.

Identifiers and local construction. The role of identifiers is different in local algorithms that need to *construct* a solution. From the perspective of construction tasks, it is easy to see that the usual $\mathcal{LOCAL}$ model is much stronger than the Id-oblivious model: there are many tasks that are trivial in $\mathcal{LOCAL}$ and impossible to solve with an Id-oblivious algorithm (examples: finding an orientation of the edges; 2-colouring a 1-regular graph).

Therefore to ask meaningful questions related to the role of unique identifiers in construction tasks, we usually compare the $\mathcal{LOCAL}$ model with models that retain some symmetry-breaking information—two such models are OI, *order-invariant algorithms*, and PO, *port numbering and orientation*.

– In the OI model [18], the output of an algorithm is not allowed to change if we reassign the identifier while preserving their relative order.
– In the PO model [17], there is an ordering on the incident edges, and all edges carry an orientation.

Note that model OI is stronger than the Id-oblivious model: in the Id-oblivious model, $A^*(G, \mathrm{Id}, v) = A^*(G, \mathrm{Id}', v)$ for *any* two assignments $\mathrm{Id}, \mathrm{Id}' \colon V(G) \to \mathbb{N}$, while in the OI model, we only require this for assignments $\mathrm{Id}, \mathrm{Id}' \colon V(G) \to \mathbb{N}$ that satisfy $\mathrm{Id}(u) < \mathrm{Id}(v) \iff \mathrm{Id}'(u) < \mathrm{Id}'(v)$. This difference makes the OI model much stronger.

Indeed, it turns out that from the perspective of strictly local algorithms, for many graph problems models $\mathcal{LOCAL}$ and OI are equally strong: Naor and Stockmeyer [18] prove that for problems whose *decision* version can be solved locally, *construction* is possible in $\mathcal{LOCAL}$ if and only if it is possible in OI. More recently, Göös et al. [9] shows that there is also a general class of *optimisation* problems for which $\mathcal{LOCAL}$, OI and PO are equally expressive.

The results of Naor and Stockmeyer [18] and Göös et al. [9] focus on bounded-degree graphs. They also make a subtle technical assumption: each node produces a local output from a constant-size set. This is necessary: Hasemann et al. [10] give an example of a natural problem that violates this assumption—and separates $\mathcal{LOCAL}$ and OI.

Bounds on n. It turns out that in decision problems, unique identifiers are helpful for one reason, and for one reason only: obtaining an estimate on n, the number of nodes. Indeed, by prior work we already know that $\mathsf{LD}^* = \mathsf{LD}$ holds assuming that every node knows an upper bound on the total number of nodes in the input graph [5].

Of course we can interpret a decision problem as a very special kind of construction problem, and therefore the present work also shows that some construction problems can exploit numerical identifiers to learn about n. However, this is a highly atypical example. For classical graph problems this information does not help a local algorithm—the identifiers are typically used for local symmetry breaking and their numerical magnitude is inconsequential.

However, if we step outside the field of strictly local algorithms, it is common to *assume* that all nodes know the same upper bound on n. This is a convenient assumption that often simplifies algorithm design. Korman et al. [15] show that in many cases it is merely a convenience—the knowledge of an upper bound on n is not essential.

2 Separation under bounded identifiers

In this section we work under assumption $(\mathbf{B}, \neg\mathbf{C})$ and exhibit a locally decidable property $\mathcal{P}$ that cannot be decided with an Id-oblivious local algorithm.

Let $f \colon \mathbb{N} \to \mathbb{N}$ be such that $\mathrm{Id}(v) < f(n)$ for all $v \in V(G)$, where G is a connected input graph. The reason identifiers are useful is that they leak information about n. For example, if a node is given an identifier i, it can deduce that $n > f^{-1}(i)$, where we denote by $f^{-1}(i)$ the smallest j such that $f(j) \geq i$.

Promise problem. As an illustration, we first describe a simple promise problem in $\mathsf{LD} \setminus \mathsf{LD}^*$.

> Promise problem: The instances are labelled graphs (G, r) where G is an n-cycle and $r \in \mathbb{N}$ is a constant input label. We promise that either $n = r$ or $n = f(r)$.
>
> We have a *yes*-instance if $n = r$ and a *no*-instance if $n = f(r)$.

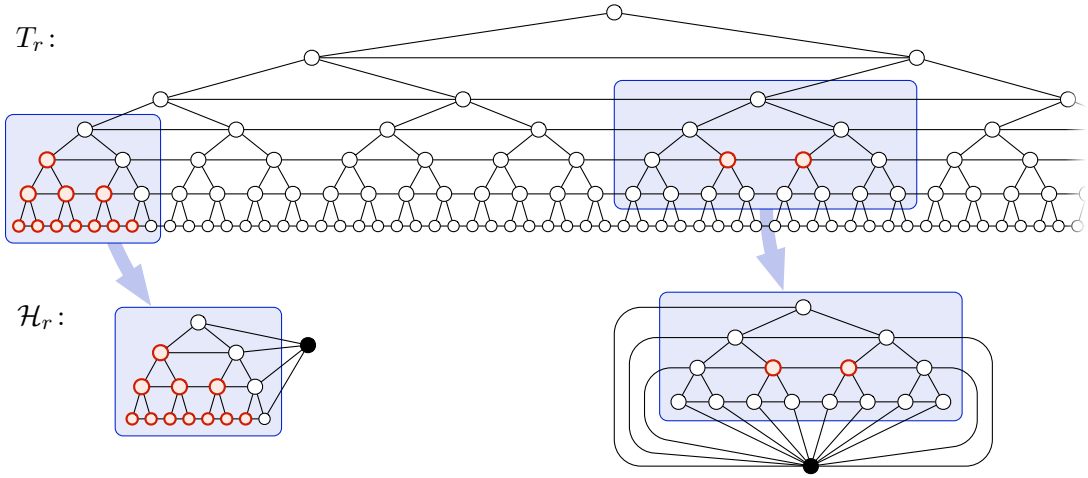

Figure 1: Graph T_r is a layered tree of depth $R(r) \gg r$. Each graph $H^+ \in \mathcal{H}_r$ is a layered tree of depth r, augmented with a single pivot node (black). The nodes that are far from the boundary (highlighted) have local neighbourhoods that are indistinguishable from the local neighbourhood of the same node in T_r.

Note that r-cycles and $f(r)$-cycles cannot be told apart by an Id-oblivious algorithm as they are locally indistinguishable topology-wise when r is large. However, we can solve the problem using identifiers: the $f(r)$-cycles can be rejected, because there is a node with identifier at least $f(r)$, which is too large to be found in the r-cycle. (We can exploit assumption ($\neg$**C**) here if f is uncomputable.)

It is not much harder to design a promise-free example in $\mathsf{LD} \setminus \mathsf{LD}^*$—we do this next.

Promise-free problem. Define $R(r) := f(2^{r+1} + 1)$. The key idea is that

- if the instance is a complete depth-r binary tree, all identifiers are smaller than $R(r)$,
- if the instance is a complete depth-$R(r)$ binary tree, there is an identifier at least $R(r)$.

Intuitively, we can use identifiers to accept "small" instances and reject "large" instances. The nontrivial part is to make sure that we can also reject instances that are neither small nor large.

A *layered depth-k tree* is a complete binary tree of depth k where, in addition, nodes at each level are connected by a path in the natural order; see Figure 1. Denote by T_r the labelled graph consisting of a layered depth-$R(r)$ tree. Each node of T_r is labelled with (r, x, y), where the coordinates (x, y) indicate the position of the node in the binary tree.

Write $H \leq_r T_r$ if a labelled graph H is an induced subgraph of the labelled graph T_r, and the topology of H is a layered depth-r tree. Call $u \in V(H)$ a *border node* if u has a neighbour in $V(T_r) \setminus V(H)$. We define H^+ to be H together with a new node (*pivot node*) that is adjacent to all the border nodes of H; see Figure 1. We collect $\mathcal{H}_r := \{H^+ : H \leq_r T_r\}$. We are now ready to define

$$\mathcal{P} := \bigcup_{r \geq 0} \mathcal{H}_r, \qquad \mathcal{P}' := \mathcal{P} \cup \{T_r : r \geq 0\}.$$

We will refer to labelled graphs in $\mathcal{P}$ as "small" instances and graphs in $\mathcal{P}' \setminus \mathcal{P}$ as "large" instances. Of course instances of $\mathcal{P}$ are only small in comparison with the parameter r

that is encoded in the labelling of the graph; we have arbitrarily large graphs in both $\mathcal{P}$ and $\mathcal{P}'$.

We will next show that the construction satisfies the following properties:

- $\mathcal{P}' \in \mathsf{LD}^*$, that is, even if we do not have access to unique identifiers, we can verify that the input is *either* small *or* large. Hence we do not need to rely on a promise—we can locally verify it.
- $\mathcal{P} \in \mathsf{LD}$, that is, we can reject large instances with the help of identifiers,
- $\mathcal{P} \notin \mathsf{LD}^*$, that is, we cannot distinguish between small and large instances with Id-oblivious algorithms.

($\mathcal{P}' \in \mathsf{LD}^*$): The overall structure of a layered depth-$R(r)$ tree is straightforward to verify locally with the help of coordinates; we can also easily check that all nodes agree on the value of r. We can verify that the coordinates satisfy $0 \le x < 2^y$ and $0 \le y \le R(r)$, there is no parent iff $y = 0$, there are no children iff $y = R(r)$, etc.

The non-trivial part is the case of a pivot node. The crucial property is that a pivot node sees *all* border nodes of a small instance. Therefore a pivot node can verify that the size of the border (as well as the coordinates of the border nodes) agree with the definition of a small instance.

In essence, if we encounter a pivot node, we must have a small instance: if we fix the structure near the border nodes, and then complete it so that it is locally consistent with the structure of a layered tree, we will arrive at a labelled graph in $\mathcal{P}$. On the other hand, if we never encounter a pivot node, we must have a large instance.

($\mathcal{P} \notin \mathsf{LD}^*$): Suppose for contradiction that A^* is a t-time Id-oblivious algorithm that decides $\mathcal{P}$. For a large enough $r \gg t$, we have that each t-neighbourhood in T_r is already found in one of the *yes*-instances in $\mathcal{H}_r$. But because A^* accepts all of $\mathcal{H}_r$, it must also accept the *no*-instance T_r, which is a contradiction.

($\mathcal{P} \in \mathsf{LD}$): The only difficulty in locally deciding $\mathcal{P}$ is to be able to reject T_r while accepting all graphs in $\mathcal{H}_r$. But there is a node in T_r with an identifier at least $R(r)$, which is too large to be found in the graphs $\mathcal{H}_r$.

3 Separation under computability

In this section we assume that all local algorithms are computable (**C**). We will exhibit a locally decidable property $\mathcal{P}$ that cannot be decided by an Id-oblivious local algorithm.

Promise problem. Again, to illustrate our approach, we first describe a simple promise problem that separates LD^* and LD.

> Promise problem $\mathcal{R}$: The instances are labelled graphs (G, M) such that G is an n-cycle; the constant input label M is a Turing machine; and if M halts in exactly s steps (when started on a blank tape) then we promise that $n \ge s$.
>
> We have a *yes*-instance if M runs forever and a *no*-instance if M halts.

($\mathcal{R} \in \mathsf{LD}$): The problem $\mathcal{R}$ is locally decidable using identifiers. Indeed, a node with identifier i first simulates M for i steps. Then, if M stops within this many steps, we output *no*; otherwise we output *yes*. For correctness, note that our promise implies that for every *no*-instance (G, M) where M halts, there will be some node v with identifier at least as large as M's run-time, and v will be able to reject (G, M).

($\mathcal{R} \notin \mathsf{LD}^*$): On the other hand, it is easy to see that any Id-oblivious algorithm for $\mathcal{R}$ has to solve the halting problem without the additional knowledge of M's run-time, which is an uncomputable task.

In this section, our goal is to construct a promise-free version of this decision problem.

3.1 Overview

The computationally difficult part in our decision problem $\mathcal{P}$ will be to determine whether a given Turing machine M halts and outputs 0 (when started on a blank tape).

To make $\mathcal{P}$ easy for an algorithm using identifiers, we will require that the instance G contains a grid-like locally checkable execution table of M. This way—as in the promise problem example—there will be some node v that has an identifier larger than M's run-time. The node v can then locally simulate M to discover its output.

To make $\mathcal{P}$ hard for an Id-oblivious algorithm, we need to obfuscate the structure of G so that its local topology does not reveal any useful information about the execution of M. In particular, even if M halts, no local neighbourhood of G should certify this fact. This way, an Id-oblivious algorithm is left with trying to find out M's output without any additional means. More formally, such an algorithm would need to separate the languages

$$L_i := \{M : M \text{ outputs } i\}, \qquad i = 0, 1,$$

which is known to be impossible for a computable function:

Lemma 1 (e.g. [19, p. 65])**.** *The languages L_0 and L_1 are* computably inseparable*, i.e., there is no computable set R such that $L_0 \subseteq R$ and $L_1 \cap R = \varnothing$.* □

Implementation. For a pair (M, r), where M halts and $r \in \mathbb{N}$ is a locality parameter, we will construct a graph $G(M, r)$ satisfying the following properties.

(P1) The execution table of M is contained in $G(M, r)$.
(P2) It is locally decidable (even in LD^*) whether an instance is of the form $G(M, r)$.
(P3) The r-neighbourhoods of $G(M, r)$ reveal only computable information about M. More formally, there is an algorithm B that halts on all inputs (N, r), where N is any Turing machine, and outputs a finite set of r-neighbourhoods $B(N, r)$ such that

$$N \text{ halts} \quad \Longrightarrow \quad B(N, r) = \{\ r\text{-neighbourhoods of } G(N, r)\ \}.$$

Note, especially, that B halts even if N does not!

Suppose for a moment that we have a construction satisfying (P1–P3). We can now define

$$\mathcal{P} := \{G(M, r) : M \text{ outputs } 0\}.$$

Theorem 2. $\mathcal{P} \in \mathsf{LD} \setminus \mathsf{LD}^*$ *under* (**C**).

Proof. ($\mathcal{P} \in \mathsf{LD}$): Given (G, Id) as input, a node $v \in V(G)$ computes in two stages. First, v performs its local test according to (P2) to see if $G = G(M, r)$ for some (M, r). If this test fails, v outputs *no*. Otherwise v proceeds to the second stage where v locally simulates M for $\mathrm{Id}(v)$ steps. If the simulation finishes and M outputs something other than 0, then v outputs *no*; otherwise v outputs *yes*.

For correctness, we need only note that in case all nodes pass the first stage, we have that $G = G(M, r)$, and thus, by (P1), there will be some node v with so large an identifier that v will finish the simulation of M in the second stage and discover M's true output.

($\mathcal{P} \notin \mathsf{LD}^*$): For the sake of contradiction, suppose that an Id-oblivious algorithm A^* with run-time t decides $\mathcal{P}$. We show how A^* can be exploited to separate the languages L_0 and L_1.

Separation algorithm R: Given a Turing machine N we first compute $B(N, t)$. Then, we run A^* on all the t-neighbourhoods in $B(N, t)$. We accept N precisely if A^* accepts all of $B(N, t)$.

First, note that, by (P3), our algorithm R halts on every input N. Moreover, suppose that N halts. Then R accepts N iff A^* accepts every t-neighbourhood of $G(N, r)$ iff A^* accepts $G(N, r)$ iff $G(N, r) \in \mathcal{P}$ iff N outputs 0. But this contradicts Lemma 1. □

Indeed, it remains to give the details of a construction satisfying (P1–P3).

3.2 Construction of $G(M, r)$

Let M be a Turing machine that halts. Each node in the graph $G = G(M, r)$ will have (M, r) as part of their input labelling. The graph G will consist of two parts:

- the *execution table* T of M, and
- a certain *fragment collection* $\mathcal{C}$.

See Figure 2.

Execution table. Let s be the running time of M. The execution table T of M will be represented, as per usual, as a labelled square grid graph on nodes $[s + 1] \times [s + 1]$, where two nodes are adjacent if their Euclidean distance is 1. We think of the edges of T as being oriented from top to bottom and from left to right. Such an orientation can be locally supplied by labelling (x, y) with (x mod 3, y mod 3).

Labels for execution. The i-th row of T corresponds to the configuration of M before the i-th step of the execution: the nodes are labelled with tape cell contents, and the read-write head of the machine is owned by exactly one node per row; this node also records the state of the machine. The first row contains just blank symbols, and the computation starts with the head on the leftmost node, which we call the *pivot node*.

The exact details of this labelling scheme are not important. Any reasonable scheme will do. We only require that the size of the labels is bounded by a computable function of M. For example, we cannot allow the nodes on the i-th row to hold the number i in their labels, since, intuitively, this would leak information about M's run-time to an Id-oblivious algorithm. (More precisely, this would mess up our construction of $\mathcal{C}$ below.)

Local decidability. It is well known that valid executions of a Turing machine can be checked locally—at least once we somehow know that the instance is really a labelled square grid and not, e.g., a torus-like graph that locally looks like a grid. To make T locally checkable, we need to augment it with some special structure; we take care of this technicality in Appendix A.

Fragment collection. The purpose of the fragment collection $\mathcal{C}$ is to ensure property (P3).

Intuition. If we had $G = T$, an Id-oblivious algorithm could decide whether M output 0 simply by checking if there was a local neighbourhood in $G = T$ where M is in a halting state with output 0.

To prevent this from happening, we add superfluous table fragments to G. In fact, we will let G contain all syntactically possible execution table fragments. This way, the answer to the question "Does there exists a local neighbourhood in G where M is in such-and-such a state" will always be *yes*. In effect, when an Id-oblivious algorithm

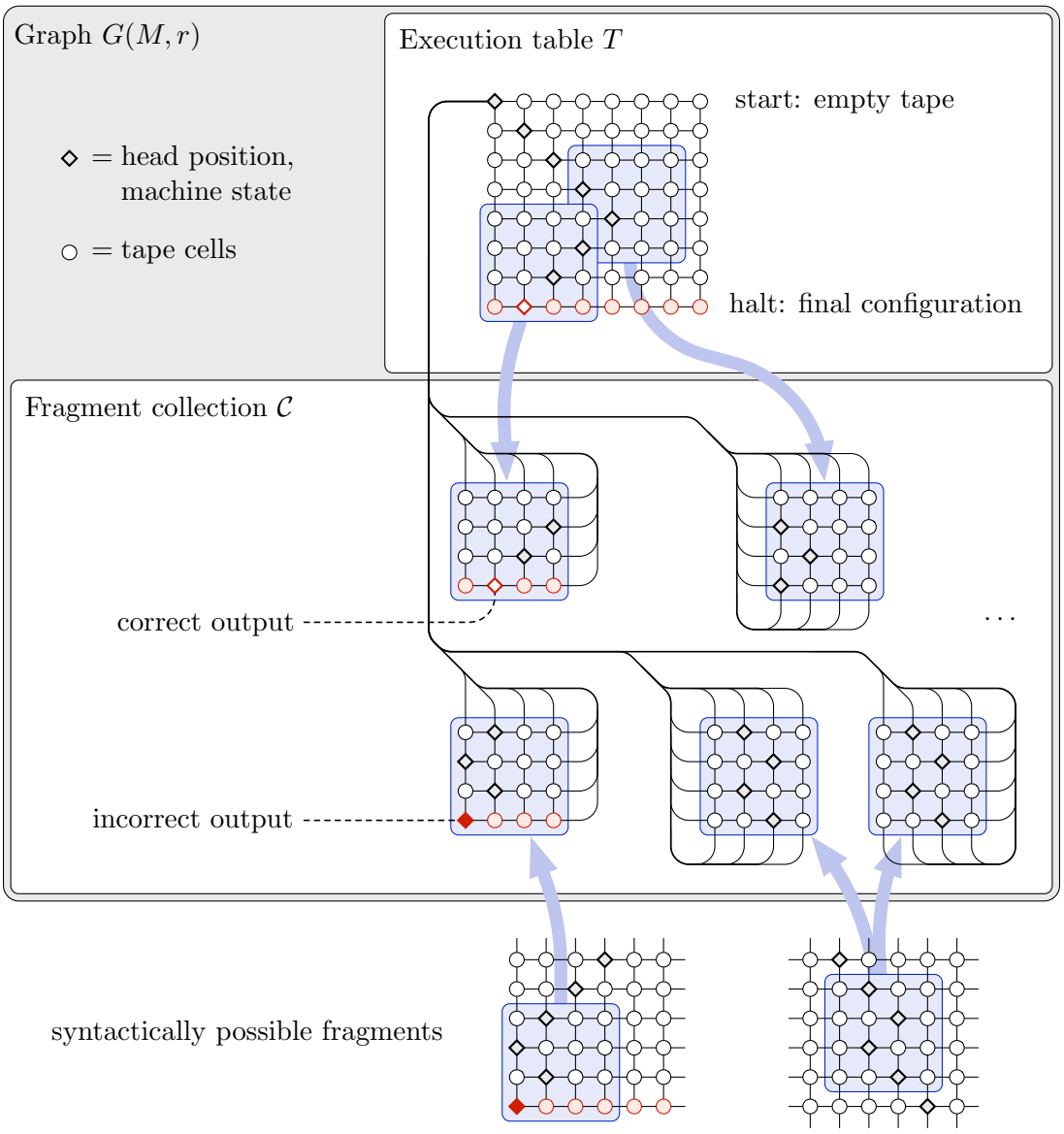

Figure 2: Construction of graph $G(M, r)$.

is exploring G locally, it learns nothing about the execution of M that it could not compute by itself.

Construction. Let F be a $3r \times 3r$ grid graph. Consider labelling F in all possible ways that satisfy the local consistency rules of T. That is, we put no limitations on how the boundary nodes are labelled, as long as

- the (mod 3)-labels give a consistent orientation, and
- every 2×2 sub-table of F is consistent with the transition function of M.

We let $\mathcal{C} = \mathcal{C}(M, r)$ consist of these labelled versions of F.

The important property here is that every r-neighbourhood in T (including those near a boundary of T) is found already in some labelled fragment in $\mathcal{C}$.

Efficiency. The construction of $\mathcal{C}$ is purely syntactic: for any machine N (that does not necessarily halt), we can efficiently generate $\mathcal{C}(N, r)$ by a simple enumeration of all possible labellings, as our labelling scheme uses bounded labels. We record this observation.

Lemma 2. *There is an algorithm that on input (N, r) outputs the finite collection $\mathcal{C}(N, r)$.* □

Putting G together. To construct G we glue together T and the fragments $\mathcal{C}$. Details follow.

Natural borders. Consider the leftmost column of nodes C in a labelled fragment $F \in \mathcal{C}$. We call C a *natural border* if C could, in principle, appear on the leftmost column of an execution table of M, i.e., if the machine head never moves to, or appears from, the left of C. We say that the rightmost column is *natural* under analogous circumstances. The bottom row is *natural* if it does not contain the machine head in a non-halting state. The top row is never natural.

Here is a technical point: we need the non-natural borders to always form a connected subgraph of F. The only situation where this is currently violated is when precisely the top and bottom rows of F are non-natural, but this is easily fixed by replacing F with two of its variants where the left and right borders are interpreted non-natural in turn. We now gain the following property, which becomes useful when proving that G is locally decidable.

> Border property: Given a subgraph induced on the non-natural borders of a fragment $F \in \mathcal{C}$, the local transition rules of M reconstruct F uniquely.

Construction. The graph G consists of (i) the table T, (ii) the fragments $\mathcal{C}$, and also (iii) new edges that connect each node of a non-natural border in $\mathcal{C}$ to the pivot node of T.

This completes the description of G. We leave the straightforward but tedious details of checking that G is locally decidable to Appendix A.

Efficiency. Finally, for the purposes of (P3), we note that our construction of $G(M, r)$ is highly explicit in the sense that the set of r-neighbourhoods of $G(M, r)$ can be computed even without the knowledge of M halting.

> Neighbourhood generator B: On input (N, r), where N does not necessarily halt, we first compute $\mathcal{C} = \mathcal{C}(N, r)$ using Lemma 2. Then, we begin constructing the (possibly infinite) computation table T of N for some $4r$ rows, each of width $4r$; call the resulting table fragment $T_{4r} \subseteq T$. We then glue $\mathcal{C}$ to the pivot of T_{4r} as described above to obtain a graph G_{4r}. Finally, we output the set of r-neighbourhoods in G_{4r} that do not contain nodes from the bottom row of T_{4r}.

The correctness of B follows from the observation that, if N halts, every r-neighbourhood in $G(N, r)$ is already found in G_{4r}. This establishes property (P3) and completes our proof.

3.3 Randomisation helps an Id-oblivious algorithm

To conclude this section, we point to another application of our property $\mathcal{P}$, this time in the setting of *randomised* local decision. Namely, we observe that $\mathcal{P}$ can be decided by an Id-oblivious algorithm if and only if we allow randomness.

A *randomised* local algorithm has access to an unbounded string of random bits. For $p, q \in (0, 1]$, we say that a randomised local algorithm A is a (p, q)-*decider* for $\mathcal{P}$ if the following holds for any input $(G, \boldsymbol{x}, \mathrm{Id})$:

- if $(G, \boldsymbol{x}) \in \mathcal{P}$, then $A(G, \boldsymbol{x}, \mathrm{Id}, v) = \mathit{yes}$ for all $v \in V(G)$ with probability at least p,
- if $(G, \boldsymbol{x}) \notin \mathcal{P}$, then $A(G, \boldsymbol{x}, \mathrm{Id}, v) = \mathit{no}$ for at least one $v \in V(G)$ with probability at least q.

The power of randomness is still lacking a full characterisation in the context of local decision [3, 6].

Randomised Id-oblivious decider for $\mathcal{P}$. Even though an Id-oblivious algorithm cannot use randomness to generate a fresh set of globally unique identifiers without any knowledge of n, we can still generate a few large numbers with high probability. This suffices for deciding $\mathcal{P}$ without identifiers, since, in addition to (P2), we only need some node v to obtain a number $n_v \geq n$ so that v can finish simulating M in n_v steps.

To this end, we let a node v toss a coin repeatedly until a head occurs, say after ℓ_v tosses. We set $n_v := 4^{\ell_v}$. The probability that no node has $n_v \geq n$ is then

$$\Pr[\,\forall v\colon n_v < n\,] \leq (1 - 1/\sqrt{n})^n = o(1).$$

That is, with probability at least $1 - o(1)$ we can reject an instance $G(M, r)$ where M halts with output other than 0. Hence, we obtain an Id-oblivious $(1, 1 - o(1))$-decider for $\mathcal{P}$.

This proves Corollary 1.

References

[1] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5): 1235–1265, 2012. doi:10.1137/11085178X.

[2] Pierre Fraigniaud and Andrzej Pelc. Decidability classes for mobile agents computing. In *Proc. 10th Latin American Symposium on Theoretical Informatics (LATIN 2012)*, volume 7256 of *LNCS*, pages 362–374, Berlin, 2012. Springer. doi:10.1007/978-3-642-29344-3_31.

[3] Pierre Fraigniaud, Amos Korman, and David Peleg. Local distributed decision. In *Proc. 52nd Symposium on Foundations of Computer Science (FOCS 2011)*, Los Alamitos, 2011. IEEE Computer Society Press. doi:10.1109/FOCS.2011.17.

[4] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Locality and checkability in wait-free computing. In *Proc. 25th Symposium on Distributed Computing (DISC 2011)*, volume 6950 of *LNCS*, pages 333–347, Berlin, 2011. Springer. doi:10.1007/978-3-642-24100-0_34.

[5] Pierre Fraigniaud, Magnus M. Halldorsson, and Amos Korman. On the impact of identifiers on local decision. In *Proc. 16th Conference on Principles of Distributed Systems (OPODIS 2012)*, LNCS, Berlin, 2012. Springer. To appear.

[6] Pierre Fraigniaud, Amos Korman, Merav Parter, and David Peleg. Randomized distributed decision. In *Proc. 26th Symposium on Distributed Computing (DISC 2012)*, volume 7611 of *LNCS*, pages 371–385, Berlin, 2012. Springer. doi:10.1007/978-3-642-33651-5_26.

[7] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Universal distributed checkers and orientation-detection tasks. Submitted, 2012.

[8] Mika Göös and Jukka Suomela. Locally checkable proofs. In *Proc. 30th Symposium on Principles of Distributed Computing (PODC 2011)*, pages 159–168, New York, 2011. ACM Press. doi:10.1145/1993806.1993829.

[9] Mika Göös, Juho Hirvonen, and Jukka Suomela. Lower bounds for local approximation. In *Proc. 31st Symposium on Principles of Distributed Computing (PODC 2012)*, pages 175–184, New York, 2012. ACM Press. doi:10.1145/2332432.2332465.

[10] Henning Hasemann, Juho Hirvonen, Joel Rybicki, and Jukka Suomela. Deterministic local algorithms, unique identifiers, and fractional graph colouring. In *Proc. 19th Colloquium on Structural Information and Communication Complexity (SIROCCO 2012)*, volume 7355 of *LNCS*, pages 48–60, Berlin, 2012. Springer. doi:10.1007/978-3-642-31104-8_5.

[11] Liah Kor, Amos Korman, and David Peleg. Tight bounds for distributed MST verification. In *Proc. 28th Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *LIPIcs*, pages 69–80, Dagstuhl, 2011. Schloss Dagstuhl. doi:10.4230/LIPIcs.STACS.2011.69.

[12] Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007. doi:10.1007/s00446-007-0025-1.

[13] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010. doi:10.1007/s00446-010-0095-3.

[14] Amos Korman, Shay Kutten, and Toshimitsu Masuzawa. Fast and compact self stabilizing verification, computation, and fault detection of an MST. In *Proc. 30th Symposium on Principles of Distributed Computing (PODC 2011)*, pages 311–320, New York, 2011. ACM Press. doi:10.1145/1993806.1993866.

[15] Amos Korman, Jean-Sébastien Sereni, and Laurent Viennot. Toward more localized local algorithms: removing assumptions concerning global knowledge. In *Proc. 30th Symposium on Principles of Distributed Computing (PODC 2011)*, pages 49–58, New York, 2011. ACM Press. doi:10.1145/1993806.1993814.

[16] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

[17] Alain Mayer, Moni Naor, and Larry Stockmeyer. Local computations on static and dynamic graphs. In *Proc. 3rd Israel Symposium on the Theory of Computing and Systems (ISTCS 1995)*, pages 268–278, Piscataway, 1995. IEEE. doi:10.1109/ISTCS.1995.377023.

[18] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

[19] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[20] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, 2000.

A Construction details

In this appendix we present the details that were skipped in Section 3.2.

Pyramidal execution table. We describe how to augment the execution table T of M so that it becomes locally checkable. For clarity of exposition, we assume that $s+1$ is a power of 2, say $s+1=2^h$ for some h—this assumption is easy to remove by modifying the following constructions slightly.

Denote the node set of T by $[2^h]\times[2^h]\times\{0\}$. We use an idea from Section 2: we attach a pyramid-shaped *layered quadtree* on top of T. That is, let $\widehat{T}$ be the graph that is arranged in layers $z=0,1,\dots,h$ such that T makes up the 0-th level; the z-th level contains a square grid on nodes $[2^{h-z}]\times[2^{h-z}]\times\{z\}$; and each node (x,y,z) on level $z\le h-1$ is connected to $(\lceil x/2\rceil,\lceil y/2\rceil,z+1)$ on level $z+1$; see Figure 3. The new nodes $V(\widehat{T})\setminus V(T)$ do not receive labels, except, of course, the universal label (M,r).

Pyramidal fragments. Since our construction is now going to use the pyramidal $\widehat{T}$ instead of T, we need to adjust our definition of the table fragments $\mathcal{C}$ accordingly. Analogously, we consider the pyramidal versions the fragments in $\mathcal{C}$:

$$\widehat{\mathcal{C}} := \{\widehat{F} : F\in\mathcal{C}\}.$$

However, since attaching a pyramid on top of a fragment decreases shortest-path distances between nodes, we need to use larger fragments than in Section 3.2. To fool

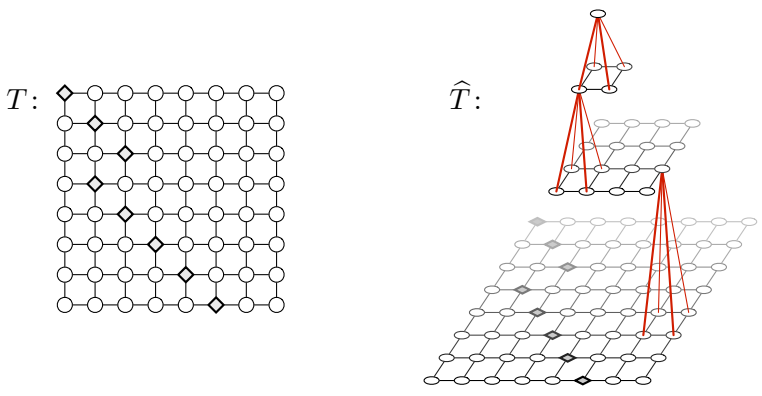

Figure 3: Table T and pyramid $\widehat{T}$.

an r-time algorithm, it is sufficient that the pyramids $\widehat{F}$ have height $3r$ (i.e., grid-size is $2^{3r} \times 2^{3r}$). This way we recover the critical property: each r-neighbourhood that could syntactically arise in $\widehat{T}$ can already be found in $\widehat{\mathcal{C}}$.

The graph $G(M, r)$ is then defined similarly as in Section 3.2: we glue the fragments $\widehat{\mathcal{C}}$ to the pivot of $\widehat{T}$ by their non-natural borders.

Note also that in verifying the property (P3) we now need the neighbourhood generator B to first construct a sub-table $T_R \subseteq T$ containing some $R = 2^{4r}$ initial rows and columns, and then glue $\widehat{\mathcal{C}}$ and $\widehat{T}_R$ together.

$G(M, r)$ is locally decidable. Suppose we are given an instance G; we argue how to locally decide (even in LD^*) whether $G = G(M, r)$ for some (M, r).

1. All nodes first make sure they are given the same pair (M, r) as part of their local input.

2. Each node in G should then belong to a layered quadtree. By design, the structure of a quadtree is such that the nodes can locally tell apart adjacent layers and recognise the inter-layer edges. In particular, each pyramid has a unique top node, which fixes its global structure.

   If the general quadtree structure is consistent, we can ignore all but the bottom-most layer of each pyramid, and be convinced that G consists of square grids that are connected together by some inter-grid edges.

3. The labelling inside each grid should follow the local execution rules of M. Also, we should have a consistent orientation on each grid.

4. The border nodes of a grid can collectively verify that the grid is either *fragment-like* (all nodes in the topmost row are incident to inter-grid edges) or a full execution table (the top-left node is the only node incident to inter-grid edges).

5. All top-left grid corners should see at least one *pivot candidate* v that is part of a full execution table. But we can impose that any such v is globally unique:

   - First, v's own execution table, call it T, cannot have any other nodes with outgoing inter-grid edges assuming that all nodes in T pass steps 3 and 4.
   - Second, consider the grids $\mathcal{C}$ that adjoin v. Node v can check that each grid in $\mathcal{C}$ has fragment-like non-natural borders. In particular, we can check that the non-natural borders form a connected subgraph in each grid—if the bottom row of a grid is non-natural, it is sufficient to verify that one of the side borders is also non-natural. But then, exploiting the *Border property* from Section 3.2, v can figure out the exact structure of $\mathcal{C}$ provided the nodes in $\mathcal{C}$ have passed step 3. It follows that there are no inter-grid edges unseen by v.

   This establishes the uniqueness of v.

6. Finally, v can check that $\mathcal{C} = \mathcal{C}(M, r)$ using Lemma 2.